AB-40

APPLICATION BRIEF

32-Bit Math Routines for the 8051

RICK SCHUE
REGIONAL APPLICATIONS SPECIALIST
INDIANAPOLIS, INDIANA

October 1992

Order Number: 270530-002

32-BIT MATH ROUTINES FOR THE 8051

CONTENTS

Here are some easy to use 16- and 32-bit math routines that take the pain out of calculations such as PID loops, A/D calibration, linearization calculations and anything else that requires 32-bit accuracy.

The package is written to interface with PL/M-51. Parameters are passed as 16-bit words to the routines, which perform operations on a 32-bit "accumulator" resident in memory. The following functions are performed:

Load_16 (word_param)

Loads a 16-bit -RD into the low half of the 32-bit "accumulator", zeros upper 16 bits of accumulator.

Load_32 (word_hi,word_lo)

Loads word_hi into upper 16 bits of accumulator, word lo_into Lower 16 bits.

Low_16

Returns the lower 16 bits of the accumulator, bits 0 through 15.

Mid_16

Returns the middle 16 bits of the accumulator, bits 8 through 23.

High_16

Returns the upper 16 bits of the accumulator, bits 16 through 31.

Mul_16 (word_param)

Multiplies the 32-bit accumulator by the 16-bit word supplied, result left in accumulator.

Div_16 (word_param)

Divides the 32-bit accumulator by the 16-bit word supplied, result left in accumulator.

Add_16 (word_param)

Adds the 16-bit word supplied to the 32-bit accumulator.

Sub_16 (word_param)

Similar to Add_16 but for subtraction.

Add_32 (word_hi,word_lo)

Forms a 32-bit value for word_hi and word_lo and adds it to the accumulator.

Sub_32 (word_hi,word_lo)

Similar to Add_32 but for subtraction.

APPLICATION

Typical applications have 16-bit "input" values and produce 16-bit "output" values, but require 32-bit values for intermediate results. An example would be reading a 12-bit A/D, performing some gain and offset calculation on the raw A/D data to produce a calibrated 16 bit result. Doing this is a simple task with this math package.

```
CALL Load_16(AD_value);

CALL Add_16 (offset_value);

CALL Mul_16 (gain_factor);

/* gain is in units of 1/256 */

result = Mid_16;
```

In this example the accumulator was loaded with the raw A/D value and then the offset was applied. The gain_factor was "pre-multiplied" by 256 (8 bits), giving it a granularity of 1/256. The result was extracted from the "middle" 16 bits of the accumulator (bits 8 to 23) to account for the scaling factor of 256 introduced in the multiply step.

The package requires about 384 bytes of ROM and 30 bytes of RAM. Individual routines can be deleted to conserve RAM if they are not used.

CODE SOURCE LISTINGS

```
                              CODE SOURCE LISTINGS

NAME    Math_32_Module
;
PUBLIC     Load_16, ?Load_16?byte
PUBLIC     Load_32, ?Load_32?byte
PUBLIC     Mul_16, ?Mul_16?byte
PUBLIC     Div_16, ?Div_16?byte
PUBLIC     Add_16, ?Add_16?byte
PUBLIC     Sub_16, ?Sub_16?byte
PUBLIC     Add_32, ?Add_32?byte
PUBLIC     Sub_32, ?Sub_32?byte
PUBLIC     Low_16, Mid_16, High_16
;
Math_32_Data     SEGMENT    DATA
Math_32_Code     SEGMENT    CODE
;
RSEG    Math_32_Data
?Load_16?byte:  DS 2
?Load_32?byte:  DS 4
?Mul_16?byte:   DS 2
?Div_16?byte:   DS 2
?Add_16?byte:   DS 2
?Sub_16?byte:   DS 2
?Add_32?byte:   DS 4
?Sub_32?byte:   DS 4
OP_0:           DS 1
OP_1:           DS 1
OP_2:           DS 1
OP_3:           DS 1
TMP_0:          DS 1
TMP_1:          DS 1
TMP_2:          DS 1
TMP_3:          DS 1
;
RSEG    Math_32_Code
```

270530-1

```
Load_16:
    ;Load the lower 16 bits of the OP registers with the value supplied
    MOV     OP_3,#0
    MOV     OP_2,#0
    MOV     OP_1,?Load_16?byte
    MOV     OP_0,?Load_16?byte + 1
    RET

Load_32:
    ;Load all the OP registers with the value supplied
    MOV     OP_3,?Load_32?byte
    MOV     OP_2,?Load_32?byte + 1
    MOV     OP_1,?Load_32?byte + 2
    MOV     OP_0,?Load_32?byte + 3
    RET

Low_16:
    ;Return the lower 16 bits of the OP registers
    MOV     R6,OP_1
    MOV     R7,OP_0
    RET

Mid_16:
    ;Return the middle 16 bits of the OP registers
    MOV     R6,OP_2
    MOV     R7,OP_1
    RET

High_16:
    ;Return the high 16 bits of the OP registers
    MOV     R6,OP_3
    MOV     R7,OP_2
    RET

Add_16:
    ;Add the 16 bits supplied by the caller to the OP registers
    CLR     C
    MOV     A,OP_0
    ADDC    A,?Add_16?byte + 1      ;low byte first
    MOV     OP_0,A
    MOV     A,OP_1
    ADDC    A,?Add_16?byte          ;high byte + carry
    MOV     OP_1,A
    MOV     A,OP_2
    ADDC    A,#0                    ;propagate carry only
    MOV     OP_2,A
    MOV     A,OP_3
    ADDC    A,#0                    ;propagate carry only
    MOV     OP_3,A
    RET
```

270530-2

```
Add_32:
    ;Add the 32 bits supplied by the caller to the OP registers
    CLR     C
    MOV     A,OP_0
    ADDC    A,?Add_32?byte + 3     ;lowest byte first
    MOV     OP_0,A
    MOV     A,OP_1
    ADDC    A,?Add_32?byte + 2     ;mid-lowest byte + carry
    MOV     OP_1,A
    MOV     A,OP_2
    ADDC    A,?Add_32?byte + 1     ;mid-highest byte + carry
    MOV     OP_2,A
    MOV     A,OP_3
    ADDC    A,?Add_32?byte         ;highest byte + carry
    MOV     OP_3,A
    RET

Sub_16:
    ;Subtract the 16 bits supplied by the caller from the OP registers
    CLR     C
    MOV     A,OP_0
    SUBB    A,?Sub_16?byte + 1     ;low byte first
    MOV     OP_0,A
    MOV     A,OP_1
    SUBB    A,?Sub_16?byte         ;high byte + carry
    MOV     OP_1,A
    MOV     A,OP_2
    SUBB    A,#0                   ;propagate carry only
    MOV     OP_2,A
    MOV     A,OP_3
    SUBB    A,#0                   ;propagate carry only
    MOV     OP_3,A
    RET


Sub_32:
    ;Subtract the 32 bits supplied by the caller from the OP registers
    CLR     C
    MOV     A,OP_0
    SUBB    A,?Sub_32?byte + 3     ;lowest byte first
    MOV     OP_0,A
    MOV     A,OP_1
    SUBB    A,?Sub_32?byte + 2     ;mid-lowest byte + carry
    MOV     OP_1,A
    MOV     A,OP_2
    SUBB    A,?Sub_32?byte + 1     ;mid-highest byte + carry
    MOV     OP_2,A
    MOV     A,OP_3
    SUBB    A,?Sub_32?byte         ;highest byte + carry
    MOV     OP_3,A
    RET
```

270530-3

```
Mul_16:
    ;Multiply the 32 bit OP with the 16 value supplied
    MOV    TMP_3,#0        ;clear out upper 16 bits
    MOV    TMP_2,#0
    ;Generate the lowest byte of the result
    MOV    B,OP_0
    MOV    A,?Mul_16?byte+1
    MUL    AB
    MOV    TMP_0,A         ;low-order result
    MOV    TMP_1,B         ;high_order result
    ;Now generate the next higher order byte
    MOV    B,OP_1
    MOV    A,?Mul_16?byte+1
    MUL    AB
    ADD    A,TMP_1         ;low-order result
    MOV    TMP_1,A         ; save
    MOV    A,B             ; get high-order result
    ADDC   A,TMP_2         ; include carry from previous operation
    MOV    TMP_2,A         ; save
    JNC    Mul_loop1
    INC    TMP_3           ; propagate carry into TMP_3
Mul_loop1:
    MOV    B,OP_0
    MOV    A,?Mul_16?byte
    MUL    AB
    ADD    A,TMP_1         ;low-order result
    MOV    TMP_1,A         ; save
    MOV    A,B             ; get high-order result
    ADDC   A,TMP_2         ; include carry from previous operation
    MOV    TMP_2,A         ; save
    JNC    Mul_loop2
    INC    TMP_3           ; propagate carry into TMP_3
Mul_loop2:
    ; Now start working on the 3rd byte
    MOV    B,OP_2
    MOV    A,?Mul_16?byte+1
    MUL    AB
    ADD    A,TMP_2         ;low-order result
    MOV    TMP_2,A         ; save
    MOV    A,B             ; get high-order result
    ADDC   A,TMP_3         ; include carry from previous operation
    MOV    TMP_3,A         ; save
    ; Now the other half
    MOV    B,OP_1
    MOV    A,?Mul_16?byte
    MUL    AB
    ADD    A,TMP_2         ;low-order result
    MOV    TMP_2,A         ; save
    MOV    A,B             ; get high-order result
    ADDC   A,TMP_3         ; include carry from previous operation
    MOV    TMP_3,A         ; save
    ; Now finish off the highest order byte
    MOV    B,OP_3
    MOV    A,?Mul_16?byte+1
```

270530–4

```
MUL     AB
ADD     A,TMP_3         ;low-order result
MOV     TMP_3,A         ; save
; Forget about the high-order result, this is only 32 bit math!
MOV     B,OP_2
MOV     A,?Mul_16?byte
MUL     AB
ADD     A,TMP_3         ;low-order result
MOV     TMP_3,A         ; save
; Now we are all done, move the TMP values back into OP
MOV     OP_0,TMP_0
MOV     OP_1,TMP_1
MOV     OP_2,TMP_2
MOV     OP_3,TMP_3
RET
```

270530-5

```
Div_16:
    ;This divides the 32 bit OP register by the value supplied
    MOV     R7,#0
    MOV     R6,#0               ;zero out partial remainder
    MOV     TMP_0,#0
    MOV     TMP_1,#0
    MOV     TMP_2,#0
    MOV     TMP_3,#0
    MOV     R1,?Div_16?byte     ;load divisor
    MOV     R0,?Div_16?byte+1
    MOV     R5,#32              ;loop count
    ;This begins the loop
Div_loop:
    CALL    Shift_D             ;shift the dividend and return MSB in C
    MOV     A,R6                ;shift carry into LSB of partial remainder
    RLC     A
    MOV     R6,A
    MOV     A,R7
    RLC     A
    MOV     R7,A
    ;now test to see if R7:R6 >= R1:R0
    JC      Can_sub             ;Carry out of R7 shift means R7:R6 > R1:R0
    CLR     C
    MOV     A,R7                ;subtract R1 from R7 to see if R1 < R7
    SUBB    A,R1                ; A = R7 - R1, carry set if R7 < R1
    JC      Cant_sub
    ;at this point R7>R1 or R7=R1
    JNZ     Can_sub             ;jump if R7>R1
    ;if R7 = R1, test for R6>=R0
    CLR     C
    MOV     A,R6
    SUBB    A,R0                ; A = R6 - R0, carry set if R6 < R0
    JC      Cant_sub
Can_sub:
    ;subtract the divisor from the partial remainder
    CLR     C
    MOV     A,R6
    SUBB    A,R0            ; A = R6 - R0
    MOV     R6,A
    MOV     A,R7
    SUBB    A,R1            ; A = R7 - R1 - Borrow
    MOV     R7,A
    SETB    C               ; shift a 1 into the quotient
    JMP     Quot
Cant_sub:
    ;shift a 0 into the quotient
    CLR     C
Quot:
    ;shift the carry bit into the quotient
    CALL    Shift_Q
    ; Test for competion
    DJNZ    R5,Div_loop
    ; Now we are all done, move the TMP values back into OP
    MOV     OP_0,TMP_0
    MOV     OP_1,TMP_1
```

270530-6

```
    MOV     OP_2,TMP_2
    MOV     OP_3,TMP_3
    RET

Shift_D:
    ;shift the dividend one bit to the left and return the MSB in C
    CLR     C
    MOV     A,OP_0
    RLC     A
    MOV     OP_0,A
    MOV     A,OP_1
    RLC     A
    MOV     OP_1,A
    MOV     A,OP_2
    RLC     A
    MOV     OP_2,A
    MOV     A,OP_3
    RLC     A
    MOV     OP_3,A
    RET

Shift_Q:
    ;shift the quotent one bit to the left and shift the C into LSB
    MOV     A,TMP_0
    RLC     A
    MOV     TMP_0,A
    MOV     A,TMP_1
    RLC     A
    MOV     TMP_1,A
    MOV     A,TMP_2
    RLC     A
    MOV     TMP_2,A
    MOV     A,TMP_3
    RLC     A
    MOV     TMP_3,A
    RET

    END
```

270530–7

INTEL CORPORATION, 2200 Mission College Blvd., Santa Clara, CA 95052; Tel. (408) 765-8080

INTEL CORPORATION (U.K.) Ltd., Swindon, United Kingdom; Tel. (0793) 696 000

INTEL JAPAN k.k., Ibaraki-ken; Tel. 029747-8511

Printed in U.S.A./xxxx/0296/B10M/xx xx